# **Universidad Cenfotec**



# **Investigación Técnica.**

Inyeccion SQL



**José Alejandro Gómez Castro**

**José Andrés Ramírez Ocampo**

# **2016**

# Introducción

Con el presente documento se pretende tener una visión más clara acerca de los aspectos básicos de la teoría de Inyección SQL, que en síntesis es una técnica de ataque o método de infiltración de código que se vale de una vulnerabilidad presente en una aplicación a la hora de realizar operaciones sobre la base de datos.

Se pretende dar una breve explicación con ayuda de ejemplos para las tecnologías que aquí se presentan, para adentrarnos más y profundizar el método de Inyección SQL. Se resaltan las implicaciones e impacto que eventualmente dicho método podría tener para nuestro repositorio de datos, además de las ventajas de defender nuestro proyecto de posibles ataques.

# ¿Qué es un SQL INJECTION?

SQL Injection es una vulnerabilidad que permite a un atacante realizar consultas a una base de datos, se vale de un incorrecto filtrado de la información que se pasa a través de los campos y/o variables que usa un sitio web, es por lo general usada para extraer credenciales y realizar accesos ilegítimos. Un fallo de este tipo puede llegar a permitir ejecución de comandos en el servidor, subida y lectura de archivos, o peor aún, la alteración total de los datos almacenados.

Una vulnerabilidad de tipo SQL Injection puede ser explotada tanto a través del método GET como del método POST.

# ¿Cómo funciona?

Una inyección en la base de datos lo que provoca en que la base de datos se comporte de una manera anormal y esto puede causar alteración en los datos, que alguien mal intencionado tenga acceso a los datos de los demás clientes o información confidencial y por último que se eliminen los datos de la base de datos.

Se aprovecha de una deficiencia en el método post o en el método get. Una inyección en la base de datos es básicamente insertar una sentencia sql en alguno de los campos de texto en la base de datos esto hace que la base de datos se comporte de forma.

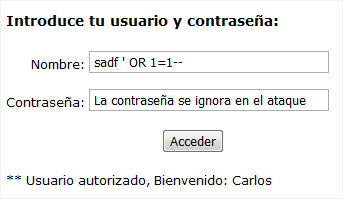
# ¿Cómo evitarlo?

El principal problema de estos ataques es que si dejamos que el usuario del programa introduzca libremente caracteres sin control ninguno (mediante formularios, por ejemplo) puede llegar a aprovecharse de las comillas (simples y dobles con las que declaramos cadenas de texto o strings).

Se pueden evitar estos ataques en muchos lenguajes distintos, e incluso hay lenguajes que por defecto hay que complicarse para que exista este fallo de seguridad, pero lo que tenemos que saber es que donde hay una consulta SQL puede haber una brecha de seguridad.

Ejemplo de una falla de seguridad:

|  |
| --- |
| SELECT \* FROM `Usuarios` WHERE `user`='' AND `pass`=''  Ejemplo de la falla con la inyección sql  SELECT \* FROM `Usuarios` WHERE `user`='' AND `pass`='' UNION SELECT \* FROM `Usuarios` WHERE `id` = '1' |



**Formas de evitar inyecciones en varios lenguajes**

**PHP**

|  |
| --- |
| $respuesta=mysql\_query("SELECT \* FROM `Usuarios` WHERE `user`='".mysql\_real\_escape\_string($name)."' AND `pass`='".mysql\_real\_escape\_string($password)."'") |

mysql\_real\_escape\_string() llama a la función mysql\_real\_escape\_string de la biblioteca de MySQL, la cual antepone barras invertidas a los siguientes caracteres: \x00, \n, \r, \, ', " y \x1a.

**.NET**

Evitar una inyección en SQL Server (con C#)

**SqlDbType.VarChar** lo que hace es convertir la variable a una literal

|  |
| --- |
| SqlConnection con = new SqlConnection(\_connectionString);  SqlCommand cmd = new SqlCommand("SELECT \* FROM Usuarios WHERE user=@user AND pass=@pass", con);  /\* Convertimos en literal estos parámetros, por lo que no podrán hacer la inyección \*/  cmd.Parameters.Add("@user", SqlDbType.VarChar, 32).Value = user;  cmd.Parameters.Add("@pass", SqlDbType.VarChar, 64).Value = password; |

También podríamos usar **AddWithValue**

|  |
| --- |
| using( SqlConnection con = (acquire connection) ) {  con. Open();  using( SqlCommand cmd = new SqlCommand("SELECT \* FROM Usuarios WHERE user=@user AND pass=@pass", con) ) {  /\* Convertimos también en literales los parámetros \*/  cmd.Parameters.AddWithValue("@user", user);  cmd.Parameters.AddWithValue("@pass", password);  using( SqlDataReader rdr = cmd.ExecuteReader() ){  /\* [...] \*/  }  }  } |

**JAVA**

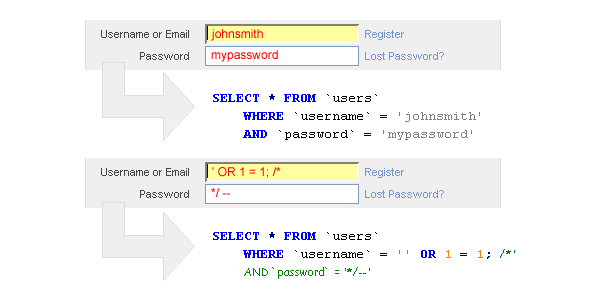
Similar a lo que se hace en C#

|  |
| --- |
| Connection con = (acquire Connection)  PreparedStatement query = con.prepareStatement("SELECT \* FROM Usuarios WHERE user=? AND pass=?");  query.setString(1, user);  query.setString(2, password);  ResultSet rset = query.executeQuery(); |

**Perl**

|  |
| --- |
| $query = $sql->prepare("SELECT \* FROM Usuarios WHERE user=? AND pass=?");  $query->execute($user,$password); |

**Ejemplos de Inyección SQL:**



Bibliografía

* <http://jbootstrap.esy.es/Investigacion-InyeccionSQL/>
* <http://www.cristalab.com/tutoriales/como-funciona-sql-injection-seguro-eres-vulnerable-c113268l/>
* <http://www.genbetadev.com/seguridad-informatica/evita-los-ataques-de-inyeccion-de-sql>
* <http://php.net/manual/es/function.mysql-real-escape-string.php>
* <http://stackoverflow.com/questions/4624811/when-should-sqldbtype-and-size-be-used-when-adding-sqlcommand-parameters>
* <http://www.w3schools.com/sql/sql_injection.asp>
* <http://tutxampp.blogspot.com/2009/11/que-es-sql-injection.html>